ONR Final Report
August 1, 1988 - September 31, 1991

Jeannette M. Wing
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

September 11, 1991

- Annual Report 1989, Maurice P. Herlihy. Incomplete.
- Annual Report 1990, Jeannette M. Wing.
- Annual Report 1991, Jeannette M. Wing.
- List of publications.

ONR Annual Report

Maurice Herlihy
School of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

September 1989

1. Numerical Productivity Measures

PI Name: Maurice Herlihy
Institution: Carnegie Mellon University
Phone Number: (617) 621-6646
E-Mail Address: Herlihy@cs.cmu.edu
Title: Research in Wait-Free Synchronization
Number: N00014-88-K-0699

- Refereed papers submitted but not yet published: 6
- Refereed papers published: 5
- Unrefereed reports and articles: 0
- Books submitted: 0
- Books published: 0
- Patents filed: 0
- Patents granted: 0
- Invited presentations: 9
- Contributed presentations: 0
- Honors: 2
- Prizes: 0
- Promotions: 0
- Graduate students: 4
- Post-docs: 0
- Minorities: 0

2. Summary of Technical Progress

PI Name: Maurice Herlihy
Institution: Carnegie Mellon University
Phone Number: (617) 621-6646
E-Mail Address: Herlihy@cs.cmu.edu
Title: Research in Wait-Free Synchronization
Number: N00014-88-K-0699

3. Detailed Summary of Technical Progress

PI Name: Maurice Herlihy
Institution: Carnegie Mellon University
Phone Number: (617) 621-6646
E-Mail Address: Herlihy@cs.cmu.edu
Title: Research in Wait-Free Synchronization
Number: N00014-88-K-0699

4. Lists of Publications, Presentations, and Reports

PI Name: Maurice Herlihy
Institution: Carnegie Mellon University
Phone Number: (617) 621-6646
E-Mail Address: Herlihy@cs.cmu.edu
Title: Research in Wait-Free Synchronization
Number: N00014-88-K-0699

References

[1] M.P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Seventh ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 276–290, August 1988.

[2] M.P. Herlihy and J.M. Wing. Reasoning about atomic objects. In *Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 193–208, September 1988.

[3] M.P. Herlihy and M.S. McKendry. Timestamp-based orphan elimination. *IEEE Transactions on Software Engineering*, 15(7):825–831, July 1989.

[4] D.L. Detlefs, M.P. Herlihy, and J.M. Wing. Inheritance of synchronization and recovery properties in Avalon/C++. *IEEE Computer*, 21(12):57–69, December 1988.

[5] M.P. Herlihy and J.M. Wing. Specifying security constraints with relaxation lattices. In *Proceedings of the Computer Security Foundations Workshop II*, pages 47–53, June 1989.

[6] M.P. Herlihy and J.D. Tygar. Implementing distributed capabilities without a trusted kernel. In *Proceedings of the International Working Conference on Dependable Computing for Critical Applications*, Santa Barbara, CA, August 1989. Reprinted in *Dependable Computing for Critical Applications*, Vol. 4 in the series "Dependable Computing and Fault-Tolerant Systems" 1991, Springer-Verlag, p. 283 - 300.

5. Transitions

PI Name: Maurice Herlihy
Institution: Carnegie Mellon University
Phone Number: (617) 621-6646
E-Mail Address: Herlihy@cs.cmu.edu
Title: Research in Wait-Free Synchronization
Number: N00014-88-K-0699

None.

6. Software and Hardware Prototypes

PI Name: Maurice Herlihy
Institution: Carnegie Mellon University
Phone Number: (617) 621-6646
E-Mail Address: Herlihy@cs.cmu.edu
Title: Research in Wait-Free Synchronization
Number: N00014-88-K-0699

None.

ONR Annual Report
October 1, 1989 - September 30, 1990

Jeannette Wing
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

September 6, 1990

1. Productivity Measure

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

- Refereed papers submitted but not yet published: 2
- Refereed papers published: 2
- Unrefereed reports and articles: 2
- Books or parts thereof submitted but not yet published: 0
- Books or parts thereof published: 0
- Patents filed but not yet granted: 0
- Patents granted: 0
- Invited presentations: 1
- Contributed presentations: 2
- Honors received (fellowships, technical society appointments, conference committee role, editorship, etc.): 2 (Herlihy)/ 3 (Wing)
- Prizes or awards received (Nobel, Japan, Turing, etc.): 0
- Promotions obtained: 1
- Graduate students supported >= 25% of full time: 7
- Post-docs supported >= 25% of full time: 0
- Minorities supported (include Blacks, Hispanics, American Indians and other native Americans such as Aleuts, Pacific Islanders, etc., Asians, and Indians): 1

2. Summary of Technical Progress

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

2.1. Summary of Goals and Expected Innovations

The goal of our research is to develop a systematic understanding of the theory and practice of highly concurrent data objects. Despite impressive progress at the hardware level that have made multiprocessor machines readily available for parallel processing, there is little agreement on the relative merits of competing architectures, and it has often proved difficult to realize these machines' potential for parallelism. Our research exploits the theory of abstract data types to derive: (1) impossibility results, showing that certain kinds of concurrency simply cannot be achieved with certain primitives; (2) new techniques for specifying and reasoning about the behavior of concurrent objects; and (3) synchronization algorithms permitting high degrees of concurrency never before achieved. The resulting theory yields consequences for algorithm and programming language design.

As yet, however, the practical implications are poorly understood. The principal objective of this research project is to undertake experimental work to translate our theoretical results into practice. Toward this end, the project's expected contributions are to contribute to these two main areas: (1) Methodology–We expect to refine our preliminary design and verification methods and apply them to more examples. (2) Experimental– We expect to build the following kinds of software: a library of concurrent data objects: a simulator that tests for incorrect behaviors of data type implementations; and a compiler that takes generates an implementation from a high-level specification. We plan to take performance measurements of our examples, comparing implementations to more standard approaches as well as comparing those running on different multiprocessor architectures.

2.2. Summary of Research Approach

We view a concurrent system as a collection of sequential processes that communicate through shared objects. Each object has a type, which defines a set of possible values and a set of primitive operations that provide the only means to create and manipulate that object. This model is general, encompassing both message-passing architectures in which the shared objects are message queues, and shared-memory architectures in which the shared objects are data structures in memory.

How does one characterize the behavior of a concurrent object? In the absence of concurrency, operations are executed one at a time, and their meanings can be captured by simple pre- and postconditions. In a concurrent program, however, operations can be executed concurrently, thus it is necessary to find a new meaning for operations that may overlap in time Two requirements seem to make intuitive sense: First, each operation should appear to "take effect" instantaneously, and second, the order of non-concurrent operations should be preserved. To capture these notions, we define a concurrent computation to be LINEARIZABLE if it is "equivalent," in a sense formally defined in to a legal sequential computation that satisfies these two requirements. We use linearizability as our fundamental correctness condition for implementations of concurrent objects.

What does it mean for an object to be highly concurrent? Traditionally, the theory of interprocess synchronization has centered around the notion of mutual exclusion: ensuring that only one process at a time is allowed to modify complex shared data objects. A variety of techniques have been proposed for implementing mutual exclusion, ranging all the way from low-level machine instructions to high-level language constructs. Nevertheless, we argue that mutual exclusion is poorly suited to multiprocessor architectures, and it is inherently incompatible with fault tolerance and real-time performance.

A novel aspect of our approach is that we do not rely on mutual exclusion as the sole means to synchronize processes. An implementation of a concurrent object is WAIT-FREE if it guarantees that any process will complete any operation within a fixed number of steps, independent of the level of contention and the execution speeds of the other processes. The wait-free condition provides fault tolerance: no process can be prevented from completing an operation by undetected halting failures of other processes, or by arbitrary variations in their speeds.

In short, our approach is based on the notions of linearizability and wait-free synchronization, which represent a qualitative break from traditional notions of interprocess synchronization.

2.3. Summary of Accomplishments

The work we did in the past year was motivated by the following observation:

> From our experience we found it difficult to design implementations of linearizable objects for arbitrary data types. Even after sketching out a possible implementation, we found that to argue its correctness we had to perform non-trivial reasoning. The complexity of the argument resulted in having to "simulate" in our heads the interleavings of some number n of processes (usually, 2 ¡= n ¡= 4), in order either (1) to convince ourselves our implementation is correct or (2) to find a contradictory behavior (sequence of interleaved operations).

Here is a summary of our major accomplishments in the areas of METHODOLOGY and EXPERIMENTAL:

2.3.1. METHODOLOGY

Design and Implementation Method:

As a compromise between relying on creative insight to implement highly concurrent objects and using the standard approach based on mutual exclusion to implement ones that could not exploit parallelism, we developed a "compilation" method that constructs non-blocking and wait-free implementa- tions of concurrent objects. This method is based on the following:

1. A LIBRARY of new synchronization and memory management algorithms, which are written for MIMD architectures in which n processes communicate by applying read, write, and compare&swap operations to shared memory. We have simulations of these algorithms.
2. The programmer chooses a representation for the data type T, and implements a set of sequential operations, written in a conventional sequential language, with no explicit synchronization. A COMPILER automatically transforms each sequential operation into a non-blocking (or wait-free) one, using the library of algorithms in (a).

After the transformation, the result is an implementation that is guaranteed to be non-blocking (or wait-free), but perhaps not the most efficient possible (had creative insight been the means of design). Hence, we avoid the problem of verification, but perhaps pay in potential concurrency.

Proof Methods

Informal

Consider the FIFO queue which has two operations, Enq and Deq. Its specification makes sense only when there is a total order relation between the items in queue, i.e., to know which is the first item in the queue. If we perform Enq and Deq operations sequentially, we get a natural total order relation on the queue's items. Suppose now we do multiple Enq operations concurrently. What total order relation can we define that will still give meaning to first? The answer is that the implementor of a concurrent object needs to define a total order relation; a proof of correctness amounts to showing that the implementation maintains this total order.

Based on our work described in our papers, we found that the most difficult part of the verification task is in defining an ordering relation in terms of the representation operations used to implement the (abstract) operations of a concurrent object, and then to argue, informally or formally, that the implementation maintains it. This ordering information is the key insight that a (human) prover must provide for each proof of correctness.

Formal

Proving formally the correctness of an implementation can be a daunting task. Although it is too early yet to assess the potential of this line of attack, we have recently begun to investigate the relevance of related work on syntax-directed proof methods for reasoning about concurrent programs. We hope to be able to develop an axiomatic proof method for establishing correctness of implementations of linearizable objects, so that one would no longer need to rely solely on simulation. The major difficulty seems to be the need to handle the potentially complex interactions of concurrent objects without recourse to brute-force analysis based on interleaving.

2.3.2. EXPERIMENTAL

Since we were indeed "simulating" in our heads the interleavings of n processes, we built a simulation environment in order to test out potential implementations. The two main components in the environment are:

1. A LIBRARY of linearizable objects, including the following linearizable data types:

- Bounded FIFO queues,
- Unbounded FIFO queues,
- Bounded priority queues,
- Unbounded priority queues,
- Semiqueues,
- Stuttering queues,
- Sets,
- "Multiple" sets,
- Read/write registers, and
- B-trees.

These implementations run on an Encore Multimax using Mach/C Threads, which provides us with lightweight processes.

2. A SIMULATOR for testing implementations of linearizable objects. There are three basic modules: *Simulate, Test, and Analyze. Simulate is the user's interface to the simulator; its main function* is, in response to the user's request, to test whether a given implementation, ConcObj, exhibits only linearizable behavior with respect to a given specification, SeqObj. The user can specify various test conditions for the simulation, e.g., the number, N, of processes to run, how long (in number of operations or time) to run each process, whether to use an input file of test cases (in the form of histories of events) or to generate a random set of test cases. Test creates N processes, and invokes concurrently on their behalf a finite number of operations on ConcObj. After all N processes terminate, the simulator stores the resulting finite concurrent history in an event list, History, and then calls the Analyze function to determine the linearizability of History. If an input file of (a finite number of) test cases is not given, Test loops, thereby generating, upon the user's request, either an infinite or a finite number of finite histories to test on a given implementation; it stops if a history is found not to be linearizable. We have run the simulator on both single and multi-processor architectures.

The analysis algorithm is exponential to handle the general case; for certain data types, e.g., FIFO queues, we have devised polynomial time analysis algorithms and are pursuing this line of study for for other types. The essence of the analysis algorithm is as follows: given a history H of a concurrent object, ConcObj, we try every possible sequential order of H's concurrent operations while preserving its real-time order relation $\rightarrow$. *We check if each sequential history Hs is linearizable by executing the operations on* SeqObj. If every possible ordering of H fails, by the definition of linearizability, the history H is non-linearizable. During this procedure of rearranging the operations of a history, we might need to undo some operations on SeqObj if we find that a tentative reordering of a subhistory of H is non-linearizable.

3. Lists of Publications, etc.

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

Refereed journal papers

M.P. Herlihy and J.M. Wing, "Linearizability: A Correctness Condition for Concurrent Objects," ACM Transactions on Programming Languages and Systems, July 1990.

Refereed conference papers

M.P. Herlihy, "A Methodology for Implementing Highly Concurrent Data Structures," Proceedings of the 2nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, March 1990.

J.M. Wing and C. Gong, "Variations on Linearizable Queues," PARLE '91, submitted September 1990.

J.M. Wing and C. Gong, "A Simulator for Concurrent Objects," Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, submitted September 1990.

Unrefereed technical reports

C. Gong and J.M. Wing, "A Library of Concurrent Objects and Their Proofs of Correctness," CMU-CS-90-151, July 1990.

J.M. Wing and C. Gong, "A Simulator for Concurrent Objects," CMU-CS-90-150, July 1990.

4. Transitions and DoD interactions

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

Industrial and university exchange

We continue to work with Dr. Maurice Herlihy who is at DEC/Cambridge Research Laboratory, Cambridge, MA. His presence at DEC/CRL provides us at CMU the perfect opportunity to exchange technical results, both ideas and software, between industry and academia. For example, we have made our code freely available to him. We expect this collaboration and technical exchange to continue for the duration of this research grant.

We also have loose connections with MIT: We implemented a linearizability B-tree data type, based on the design and proof of Prof. William Weihl and we collected a linearizable set implementation written by Kathy Yelick as part of her Ph.D. dissertation.

DoD interaction

Part of this research is additionally funded by the special joint NSF/DARPA Program on Parallel Computing Theory. This funding provides support for another faculty member at Carnegie Mellon (Prof. Stephen Brookes) and one-to-two graduate students.

5. Software and Hardware Prototypes

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

Software Prototypes

We can provide the entire research community the following two pieces of software:

- A LIBRARY of linearizable objects, including the following linearizable data types: unbounded FIFO queues, bounded FIFO queue, unbounded priority queues, bounded priority queues, stuttering queues, semiqueues, sets, multiple sets, atomic registers, and B-trees. These implementations run on an Encore Multimax using Mach/C Threads, which provides us with lightweight processes.

  We welcome additions from others in the community to add to our library.

- A SIMULATOR, which takes as input a concurrent implementation of a data type, plus user-definable parameters like number of processes to run, average number of operations to execute per process, duration to run each simulation.

Two technical reports and two papers submitted for publication document both pieces of software.

ONR Annual Report
October 1, 1990 - September 30, 1991

Jeannette Wing
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

September 11, 1991

1. Productivity Measure

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

- Refereed papers submitted but not yet published: 1
- Refereed papers published: 1
- Unrefereed reports and articles: 1
- Books or parts thereof submitted but not yet published: 0
- Books or parts thereof published: 2
- Patents filed but not yet granted: 0
- Patents granted: 0
- Invited presentations: 8
- Contributed presentations: 2
- Honors received (fellowships, technical society appointments, conference committee role, editorship, etc.): 6
- Prizes or awards received (Nobel, Japan, Turing, etc.): 0
- Promotions obtained: 0
- Graduate students supported >= 25% of full time: 7 (only 1 on this grant)
- Post-docs supported >= 25% of full time: 0
- Minorities supported (include Blacks, Hispanics, American Indians and other native Americans such as Aleuts, Pacific Islanders, etc., Asians, and Indians): 1 (summer only)

2. Summary of Technical Progress

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

2.1. Summary of Goals and Expected Innovations

The goal of our research is to develop a systematic understanding of the theory and practice of highly concurrent data objects. Despite impressive progress at the hardware level that have made multiprocessor machines readily available for parallel processing, there is little agreement on the relative merits of competing architectures, and it has often proved difficult to realize these machines' potential for parallelism. Our research exploits the theory of abstract data types to derive: (1) impossibility results, showing that certain kinds of concurrency simply cannot be achieved with certain primitives; (2) new techniques for specifying and reasoning about the behavior of concurrent objects; and (3) synchronization algorithms permitting high degrees of concurrency never before achieved. The resulting theory yields consequences for algorithm and programming language design.

As yet, however, the practical implications are poorly understood. The principal objective of this research project is to undertake experimental work to translate our theoretical results into practice. Toward this end, the project's expected contributions are to contribute to these two main areas: (1) Methodology–We expect to refine our preliminary design and verification methods and apply them to more examples. (2) Experimental– We expect to build the following kinds of software: a library of concurrent data objects; a simulator that tests for incorrect behaviors of data type implementations; and a compiler that takes generates an implementation from a high-level specification. We plan to take performance measurements of our examples, comparing implementations to more standard approaches as well as comparing those running on different multiprocessor architectures.

2.2. Summary of Research Approach

We view a concurrent system as a collection of sequential processes that communicate through shared objects. Each object has a type, which defines a set of possible values and a set of primitive operations that provide the only means to create and manipulate that object. This model is general, encompassing both message-passing architectures in which the shared objects are message queues, and shared-memory architectures in which the shared objects are data structures in memory.

How does one characterize the behavior of a concurrent object? In the absence of concurrency, operations are executed one at a time, and their meanings can be captured by simple pre- and postconditions. In a concurrent program, however, operations can be executed concurrently, thus it is necessary to find a new meaning for operations that may overlap in time. Two requirements seem to make intuitive sense: First, each operation should appear to "take effect" instantaneously, and second, the order of non-concurrent operations should be preserved. To capture these notions, we define a concurrent computation to be LINEARIZABLE if it is "equivalent," in a sense formally defined in to a legal sequential computation that satisfies these two requirements. We use linearizability as our fundamental correctness condition for implementations of concurrent objects.

What does it mean for an object to be highly concurrent? Traditionally, the theory of interprocess synchronization has centered around the notion of mutual exclusion: ensuring that only one process at a time is allowed to modify complex shared data objects. A variety of techniques have been proposed for implementing mutual exclusion, ranging all the way from low-level machine instructions to high-level language constructs. Nevertheless, we argue that mutual exclusion is poorly suited to multiprocessor architectures, and it is inherently incompatible with fault tolerance and real-time performance.

A novel aspect of our approach is that we do not rely on mutual exclusion as the sole means to synchronize processes. An implementation of a concurrent object is WAIT-FREE if it guarantees that any process will complete any operation within a fixed number of steps, independent of the level of contention and the execution speeds of the other processes. The wait-free condition provides fault tolerance: no process can be prevented from completing an operation by undetected halting failures of other processes, or by arbitrary variations in their speeds.

In short, our approach is based on the notions of linearizability and wait-free synchronization, which represent a qualitative break from traditional notions of interprocess synchronization.

2.3. Summary of Accomplishments

We continued to make progress in both the METHODOLOGY and EXPERIMENTAL areas:

Methodology : We pursued the use of Stephen Brookes's proof method for reasoning about concurrent systesm as a basis for a more formal proof method for proving linearizability. We identified why Brookes's method is inappropriate and pointed out new research directions for Brookes, giving correctness conditions like linearizability. (Brookes's method is described in "On the Axiomatic Treatment of Concurrency," CMU-CS-85-106.)

Experimental : We continued to build more linearizable objects for the library, adding the list and stack data types. As for previous implementations, we tested them using our simulator and provide informal proofs of correctness.

Experimental : We build an "registry" interface to our library so that other users can donate their implementations.

2.3.1. METHODOLOGY

While linearizability is a useful and intuitively appealing condition, to prove that a given implementation is correct is still a difficult task. A proof following our current method must be hand-tailored for each representation object and must refer to the state of all concurrent processes. We follow the Owicki-Gries style of proof, though use a informal reasoning, rather than pure syntax-directed reasoning. The key to each informal proof is in identifying a partial order among atomic instructions on the representation type that guaranteed correctness at the abstract level; then, the proof of each abstract operation entailed showing that the partial order is maintained. We discuss this problem present hand proofs of numerous examples in [2].

Rather than following an informal proof method, we explored the possibility of using Stephen Brookes's more formal proof method, which is a semantically-based axiomatic treatment of parallel programs. Brookes uses an assertion language for expressing semantic properties of concurrent program and the structure of the assertions reflects the structure of the semantic representation of a concurrent program. We thought it would be easier to express the partial order property as well as other global state information in Brookes's assertion language. We could furthermore use his composition rules on assertions (rather than on programs) to make general statements about our implementations.

We found the following limitations with Brookes's proof method as applied to our correctness condition:

1. We need to deal with two sources of nondeterminism, manifested in a computation by pending operations performed by concurrent processes. Brookes's assertion language is not appropriate for concisely expressing the inherent nondeterminism exhibited by linearizable objects.

2. We need to discuss more than one level of computation: the representation level, e.g., an array with FETCH-AND-ADD atomic instructions, as well as the abstract level, e.g., the FIFO queue with enqueue and dequeue operations. Brookes's method deals with only one level of computation.

We discuss each of these in more detail below.

The Assertion Language

The key idea underlying Brookes's assertion language is that the execution of a concurrent program Γ on initial state s can be represented by a tree structure where each node represents a part of program state (a resumption-state pair) and each branch is the execution of an atomic instruction. The tree structure suggests a class of assertions with components representing the branch structure of trees:

$$\phi ::= P \sum_{i=1}^{n} \alpha_i P_i \phi_i$$

where P and P_i are drawn from some condition language, and α_i are labels for atomic instructions.

With this assertion language, we can express the property that a program Γ satisfies an assertion ϕ by:

$$\Gamma \; sat \; \phi$$

For example, The program $[\alpha : x := x + 1 || \beta : x := x + 1]$ satisfies the assertion

$$\{x = 0\}(\alpha\{x = 1\}\{x = 1\}\beta\{x = 2\} + \beta\{x = 1\}\{x = 1\}\alpha\{x = 2\})$$

We noticed that the examples given in Brookes's paper only show how to use assertions P and P_i to express properties like $x = 3$ *and* $y = 1$; namely examples deal only with the values of variables. However such assertions are not expressive enough for proving linearizability: we need to talk about the exact position of some processes. Moreover, we need to accommodate a dynamically changing number of processes. E.g., at any given point of time, a process that is in the middle of dequeueing an element from a FIFO queue may or may not have completed its dequeue. We need a way to assert that either the dequeue has not yet completed or that it has, and that in either case, the abstract value of the FIFO queue is correct. Clearly, information about the program counters of each process can be encoded in the assertions, much like auxiliary variables are used to encode the same information in Owicki-Gries style proofs. However, Brookes's method does not help reduce the complexity of the assertion. There is a also a minor problem in Brookes's original method which assumes a fixed number (or statically determinable) number of processes.

Abstract and Representation Levels

At the abstract level, a concurrent data object's internal state is abstractly mapped to set of linearized values. For a particular data type, we define the semantics of its (abstract) operations in terms of the internal state before and after the operations. Recall that each (abstract) operation consists of two events, the invocation event and the corresponding reply event. Since several processes may apply operations to the same object concurrently, we cannot take an abstract operation as an atomic step. Instead, we take an event as an atomic step. So we have to define the semantics of an abstract operation in two steps, one for the invocation event and the other for the reply event. For event e we use $\{st\}e\{st'\}$ to denote its semantics, where st and st' are the internal states before and after e respectively. If a process just issued an invocation event for operation OP but has not received a reply event we say that OP is pending. We assume the internal state of the object has a mechanism to keep track of all those pending operations.

Given the semantics of each event, we can give the sequential specification of a history by two states, the initial and the final state:

$$(H = eH_1) \wedge (\{st\}e\{st''\}) \wedge (\{st''\}H_1\{st'\} \Rightarrow \{st\}H\{st'\}$$

Now we can use a class of assertions ϕ to specify a concurrent data object X:

- We use a condition language to represent the internal state of X. In other words, P and P_i will be of the form:
$$St = \{x | x \textit{ is a linearized value for } X\}$$
- Each node of the tree structure corresponds to an internal state and each edge (a label) corresponds to an event.

A history H thus correspond to a path in the assertion tree ϕ. We call such an assertion tree the abstract tree.

Suppose ϕ is an assertion tree for object X and the root condition is St, we say ϕ is linearizable if the following are true:

- Suppose P is a path starting from the root of ϕ and the ending state of P is st', H is the corresponding history with the following semantics
$$\{St\}H\{St''\}$$
then $St' \subseteq St''$.

To specify the implementation in Brookes's assertion language, we let each node represent a possible representation data value and each branch indicate an atomic instruction. In such a tree, a path corresponds to a sequence of atomic instructions. Here now, we must deal with the global information: knowing the number of processes that could be active at a time, and moreover, knowing what each of those processes is in the middle of doing. Since Brookes's proof system is syntax-directed, meaning that the proof is based on the text of the concurrent program itself, we do not have explicit information about these other processes: therefore, we must again resort to auxiliary variables to encode this information. As a result we are no better off using this method than using our original Owicki-Gries style of proof. To be informal, yet concrete, a typical assertion would be of the form "if process P is in the middle of operation OP and has not completed, then the abstract state of the queue is Q1; if process P is in the middle of operation OP and has completed, then the abstract state of the queue is Q2." The proof would require that in all states Q1 and Q2 are valid abstract values. The parameters in the assertion are P (not just what processes are there but a varying number), OP (what is each of those processes trying to do), completed/not yet completed, and Q (the state of the abstract objects, as represented as a set of linearized values). Hence the case explosion in the assertions makes the task of doing a proof for even a simple example (like the FIFO queue) unmanageable.

Therefore, the inherent nondeterminism in linearizability makes it awkward to use Brookes's approach. We emphasize that there is nothing wrong with Brookes's approach for the classes of programs he has investigated; we only remark that linearizability brings out limitations to traditional approaches to reasoning about concurrency. To conclude, our study raises the more interesting line of further research, orthogonal to our own interest in linearizability, for Brookes's method. For example, further work on showing a correspondence between trees at the abstract level and trees at the representation, e.g., a node at the abstract level standing for a subtree at the represnetation level, would be a first step at generalizing Brookes's technique to deal with multiple levels of computation.

2.3.2. EXPERIMENTAL

We continued to make progress on two fronts in our experimental work.

LIBRARY: Last year we started building up a library of concurrent objects and we built a simulator to use to test our implementations. This year, we added to our library implementations for linearizable lists and stacks. We tested them using our simulator and provide informal proofs of correctness (see [5]).

The implementations of the stack and list is very similar to the implementations of the other data types. We represent a stack with an array (we assume it is large enough) to hold items and an integer as top pointer. Initially we set all slots of the array to a special value *NULL* and the pointer to initial index, 0. The *Push(x)* operation first atomically gets a slot and increases the pointer; it then atomically stores x into the slot. The *Pop()* operation first atomically gets the top pointer value then scans the array from top to bottom; during the scan the *Pop* atomically SWAPs a special value *NULL* with each slot until it finds the first non-NULL value, which it then returns. This implementation is similar to the array implemenation of the FIFO queue; it differs in the order in which the array is scanned to preserve stack semantics.

The list data type provides an append operation that puts an item at the end of the list and a delete operation that removes a given item x from the list if x is there. We use a linked list as the representation and are currently adding more abstract operations to its interface.

REGISTRY: We also built an interface to our library so that others can donate implementations. We have a simple interactive registry information that records general information about each donation and organizes donated files in a common hierarchical naming scheme. We have already shared our simulator and library with Professor Kathy Yelick at University of California, Berkeley, and intend to share our software with others in the parallel programming community.

3. Lists of Publications, etc.

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

Refereed journal papers

[1] M.P. Herlihy and J.M. Wing, "Specifying Graceful Degradation,"
IEEE Transactions on Parallel and Distributed Computing,
Vol. 2, No. 1, January 1991, pp. 93-104.

Submitted journal papers

[2] J.M. Wing and C. Gong, "Testing and Verifying Concurrent Objects,"
submitted to *Journal of Parallel and Distributed Computing*,
December 1990, revision submitted August 1991.

Unrefereed technical reports

[3] M.P. Herlihy, S.-Y. Ling, and J.M. Wing,
"Implementation of Commit Timestamps in Avalon,"
CMU-CS-91-107, January 1991.

Internal Notes

[4] J.M. Wing and C. Gong, "On the Proof of Linearizability,"
on-line documentation, September 1991.

[5] J.M. Wing and C. Gong, "Proofs of Correctness for the Linearizable
Lists and Stacks," on-line documentation, September 1991.

4. Transitions and DoD interactions

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

Industrial and university exchange

At Carnegie Mellon, Professor Brian Bershad is using our implementations of linearizable objects to explore the merits of providing hardware support for atomic instructions like FETCH-AND-ADD and COMARE-AND-SWAP.

Outside of Carnegie Mellon I am collecting implementations of linearizable and wait-free objects from Dr. Maurice Herlihy, at DEC/Cambridge Research Laboratory, Prof. William Weihl, at MIT, and Prof. Kathy Yelick at Univ. of California at Berkeley. We have given Yelick access to our library of linearizable objects and our simulator. Yelick and I have agreed that if funding is available to her, she will take over the maintenance and support of our software.

DoD interaction

Part of this research is additionally funded by the special joint NSF/DARPA Program on Parallel Computing Theory. This funding provides support for another faculty member at Carnegie Mellon (Prof. Stephen Brookes) and one or two graduate students.

5. *Software and Hardware Prototypes*

PI Jeannette M. Wing

Institution Carnegie Mellon University

Phone (412) 268-3068

E-mail wing@cs.cmu.edu

Contract Title Research in Wait-Free Synchronization

Contract Number N00014-88-K-0699

Software Prototypes

We can provide the entire research community the following two pieces of software, both documented in the two technical reports, CMU-CS-90-150 and CMU-CS-90-151:

- A LIBRARY of linearizable objects, including the following linearizable data types: unbounded FIFO queues, bounded FIFO queue, unbounded priority queues, bounded priority queues, stuttering queues, semiqueues, sets, multiple sets, atomic registers, B-trees, stacks, and linked lists. These implementations run on an Encore Multimax using Mach/C Threads, which provides us with lightweight processes.

  There is a REGISTRY interface to the library for people interested in making donations of other implementations.

  We welcome additions from others in the community to add to our library.

- A SIMULATOR, which takes as input a concurrent implementation of a data type, plus user-definable parameters like number of processes to run, average number of operations to execute per process, duration to run each simulation.

List of Publications

Book chapters

1. D.L. Detlefs, M.P. Herlihy and J.M. Wing, "Avalon/C++," in *Advanced Language Implementation: Recent Research at Carnegie Mellon University*, P. Lee, editor, MIT Press, 1991

2. J.M. Wing et al., "The Avalon Language," Part IV, Chapters 19-22, in *Camelot and Avalon: A Distributed Transaction Facility*, J. Eppinger, L. Mummert and A. Spector, editors, Morgan Kaufmann Publishers, Inc., 1991.

Refereed journal papers

1. M.P. Herlihy and J.M. Wing, "Specifying Graceful Degradation,"
IEEE Transactions on Parallel and Distributed Computing,
Vol. 2, No. 1, January 1991, pp. 93-104.

2. M.P. Herlihy and J.M. Wing, "Linearizability: A Correctness Condition for Concurrent Objects," ACM Transactions on Programming Languages and Systems, July 1990.

3. M.P. Herlihy and M.S. McKendry. Timestamp-based orphan elimination. *IEEE Transactions on Software Engineering*, 15(7):825–831, July 1989.

4. D.L. Detlefs, M.P. Herlihy, and J.M. Wing. Inheritance of synchronization and recovery properties in Avalon/C++. *IEEE Computer*, 21(12):57–69, December 1988.

Submitted journal papers

1. J.M. Wing and C. Gong, "Testing and Verifying Concurrent Objects,"
submitted to *Journal of Parallel and Distributed Computing*,
December 1990, revision submitted August 1991.

Refereed conference papers

1. M.P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Seventh ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 276–290, August 1988.

2. M.P. Herlihy and J.M. Wing. Reasoning about atomic objects. In *Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 193–208, September 1988.

3. M.P. Herlihy and J.M. Wing. Specifying security constraints with relaxation lattices. In *Proceedings of the Computer Security Foundations Workshop II*, pages 47–53, June 1989.

4. M.P. Herlihy and J.D. Tygar. Implementing distributed capabilities without a trusted kernel. In *Proceedings of the International Working Conference on Dependable Computing for Critical Applications*, Santa Barbara, CA, August 1989. Reprinted in *Dependable Computing for Critical Applications*, Vol 4 in the series "Dependable Computing and Fault-Tolerant Systems" 1991, Springer-Verlag, p. 283 300.

5. M.P. Herlihy, "A Methodology for Implementing Highly Concurrent Data Structures," Proceedings of the 2nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, March 1990.

Unrefereed technical reports

1. M.P. Herlihy, S.-Y. Ling, and J.M. Wing,
"Implementation of Commit Timestamps in Avalon,"
CMU-CS-91-107, January 1991.

2. C. Gong and J.M. Wing, "A Library of Concurrent Objects and Their Proofs of Correctness," CMU-CS-90-151, July 1990.

3. J.M. Wing and C. Gong, "A Simulator for Concurrent Objects," CMU-CS-90-150, July 1990.

4. M.P. Herlihy and J.M. Wing, "Specifying Graceful Degradation," CMU-CS-88-121, March 1988.

5. M.P. Herlihy and J.M. Wing, "Linearizability: A Correctness Condition for Concurrent Objects," CMU-CS-88-120, March 1988.

6. M.P. Herlihy and J.M. Wing, "Reasoning About Atomic Objects," CMU-CS-87-176, November 1987.

7. D.L. Detlefs, M.P. Herlihy and J.M. Wing, "Inheritance of Synchronization and Recovery Properties in Avalon/C++," CMU-CS-87-133, June 1987.

8. M.P. Herlihy and J.M. Wing, "Specifying Graceful Degradation in Distributed Systems," CMU-CS-87-120, April 1987.

Unrefereed submissions and internal notes

1. J.M. Wing, "Program Specification" and "Formal Methods," *Encyclopedia of Computer Science and Technology*, A. Ralston (ed.), Van Nostrand Reinhold, to appear in *1991 edition*.

2. J.M. Wing and C. Gong, "On the Proof of Linearizability,"
on-line documentation, September 1991.

3. J.M. Wing and C. Gong, "Proofs of Correctness for the Linearizable
Lists and Stacks," on-line documentation, September 1991.